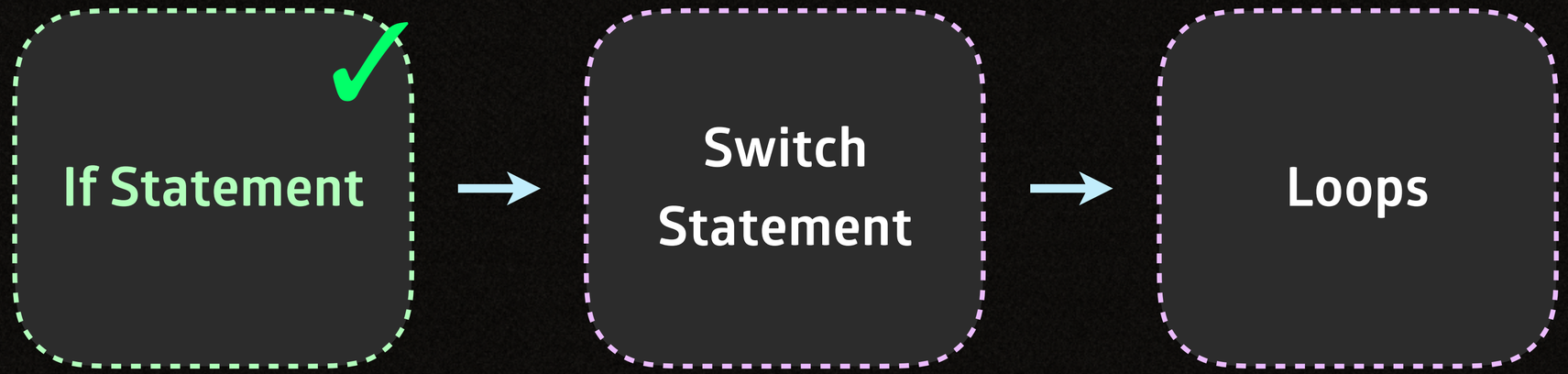
PART III

Fundamentals

Control Flow



SWITCH STATEMENT

Time to learn how to control the execution flow using the switch statement

Switch Statement → Bool Expressions → Short Switch → Parts of the Day

SWITCH STATEMENT

Similar to an if statement but with a different syntax

SWITCH STATEMENT

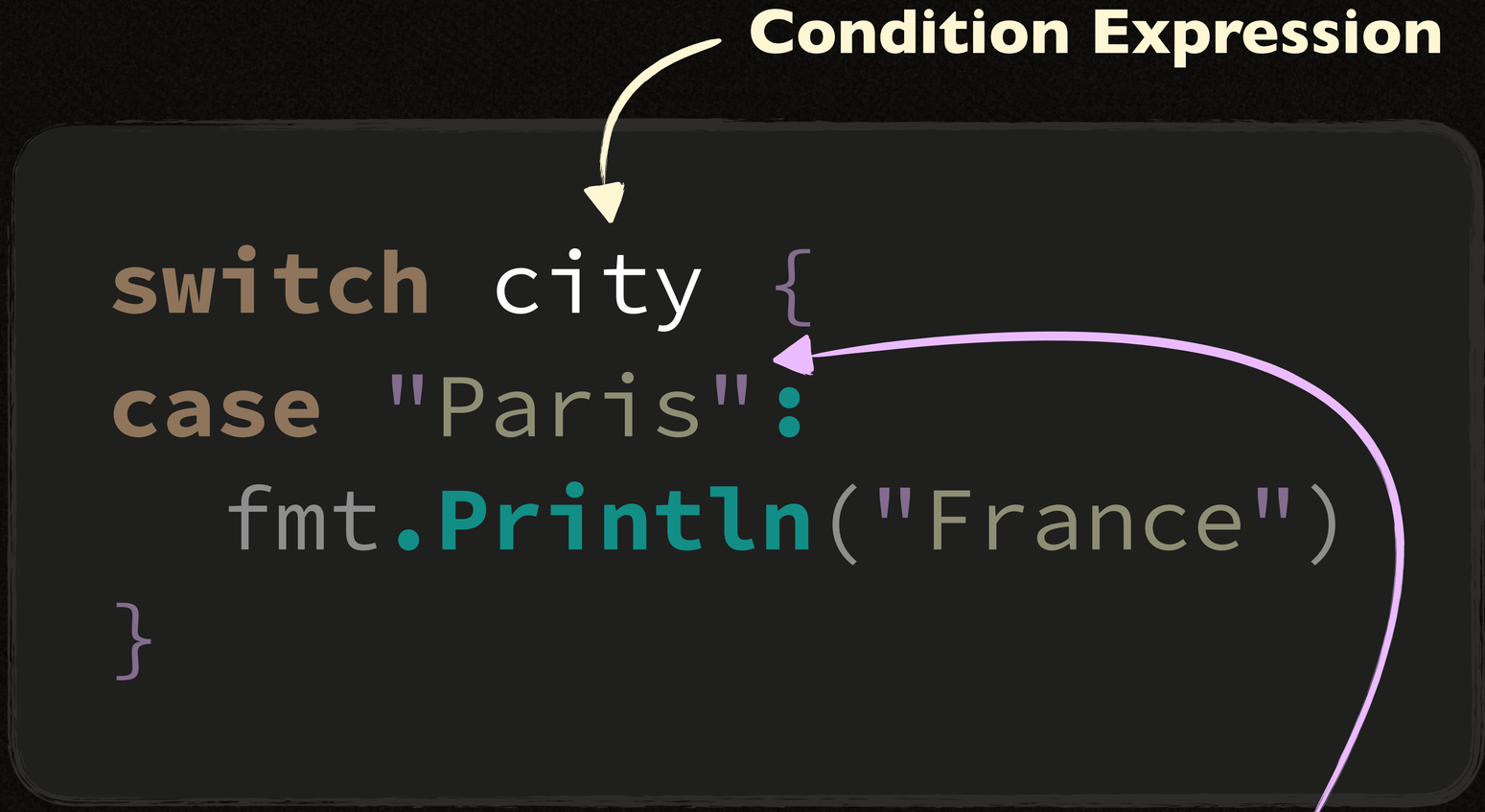
Switch statement looks for an equal value

```
city := "Paris"

switch city {
case "Paris":
   fmt.Println("France")
}
```

CONDITION EXPRESSION

Switch's condition will be compared for each case condition



CASE CLAUSE

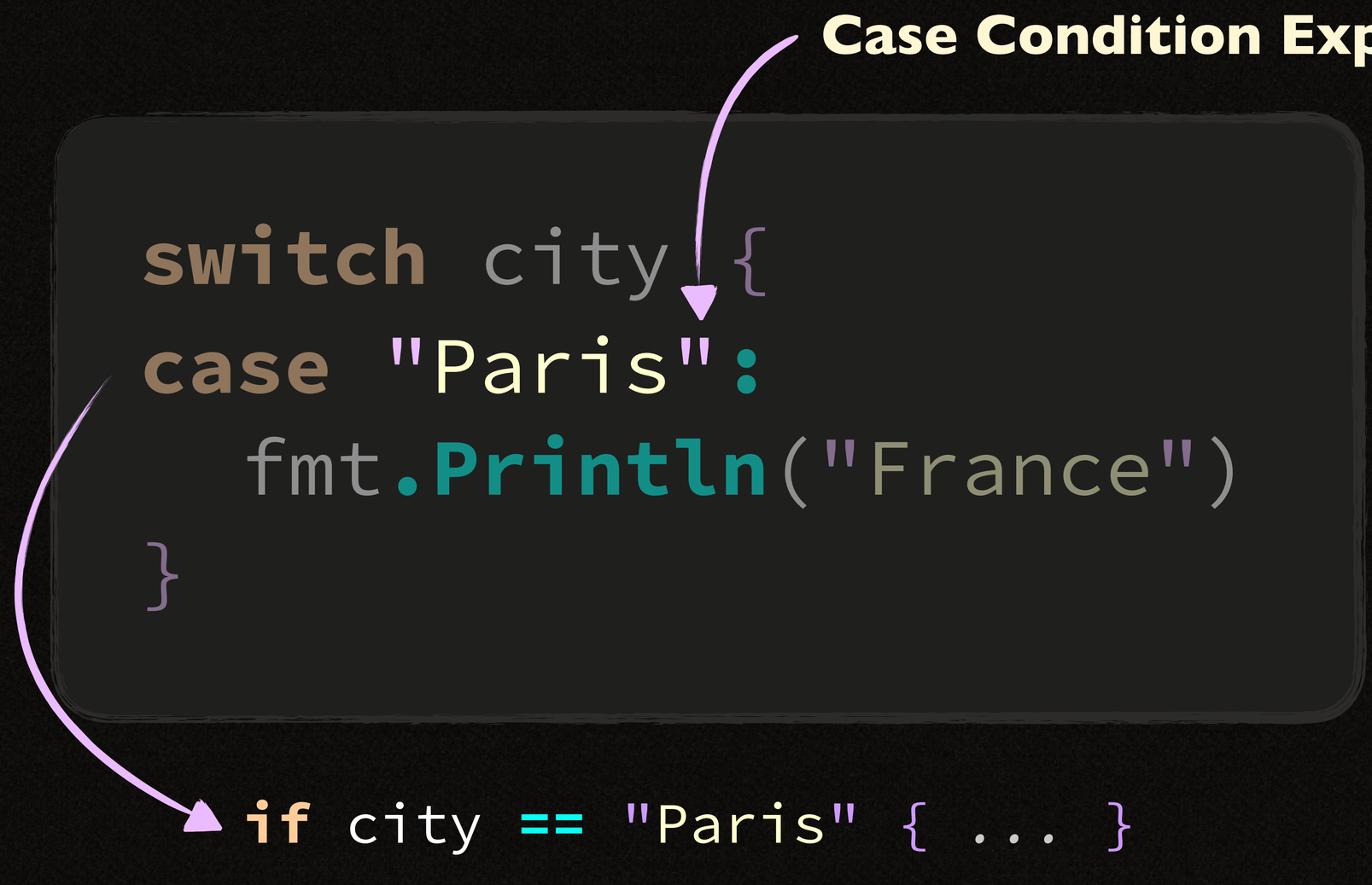
Case clause creates an exclusive block only for itself

Case Clause

```
switch city {
case "Paris":
   fmt.Println("France")
}
```

CASE CONDITION

If the city is equal to "Paris"; Go will execute the case clause's block



MULTIPLE CASES

Condition expression will be compared for each case condition

```
switch city {
case "Paris":
   fmt.Println("France")
case "Tokyo":
   fmt.Println("Japan")
}
```

MULTIPLE CASES

Condition expression will be compared for each case condition

DEFAULT CLAUSE

Similar to an if statement's else branch

DEFAULT CLAUSE

Previously...

```
city := "Paris"

switch city {
case "Paris":
   fmt.Println("France")
}
```

DEFAULT CLAUSE

The default clause is executed when no cases match

```
city := "Limbo"

switch city {
case "Paris":
   fmt.Println("France")
default:
   fmt.Println("Where?")
}
```

MULTIPLE CONDITIONS

Similar to logical OR ||

MULTIPLE CONDITIONS

Previously...

```
switch city {
case "Paris":
   fmt.Println("France")
case "Tokyo":
   fmt.Println("Japan")
}
```

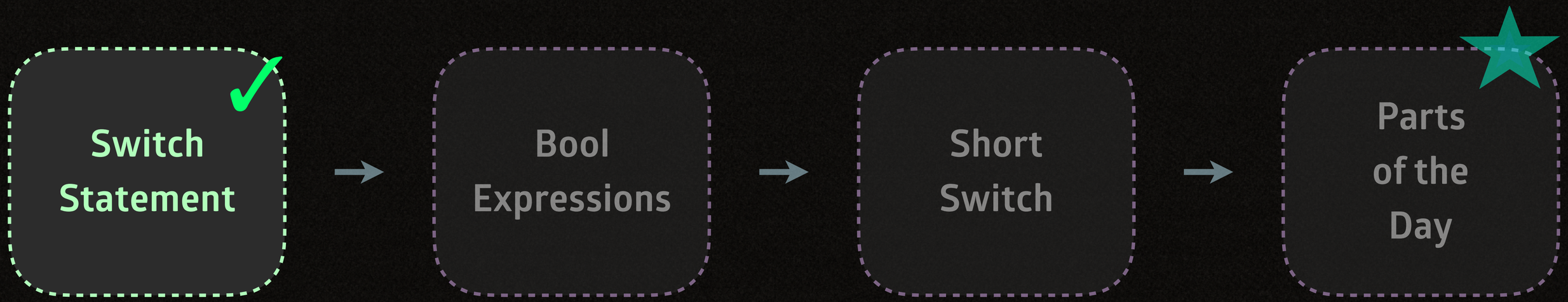
MULTIPLE CONDITIONS

You can add as many conditions as you want [keep it readable 👼 :]

```
switch city {
case "Paris", "Lyon":
   fmt.Println("France")
case "Tokyo":
   fmt.Println("Japan")
}
```

SWITCH STATEMENT

Congrats! Now you know about the basics of the switch statement!



BOOL EXPRESSIONS

You can also use boolean expressions as the conditions

BOOL EXPRESSIONS

Remember: Case conditions' types should match to condition expression's type

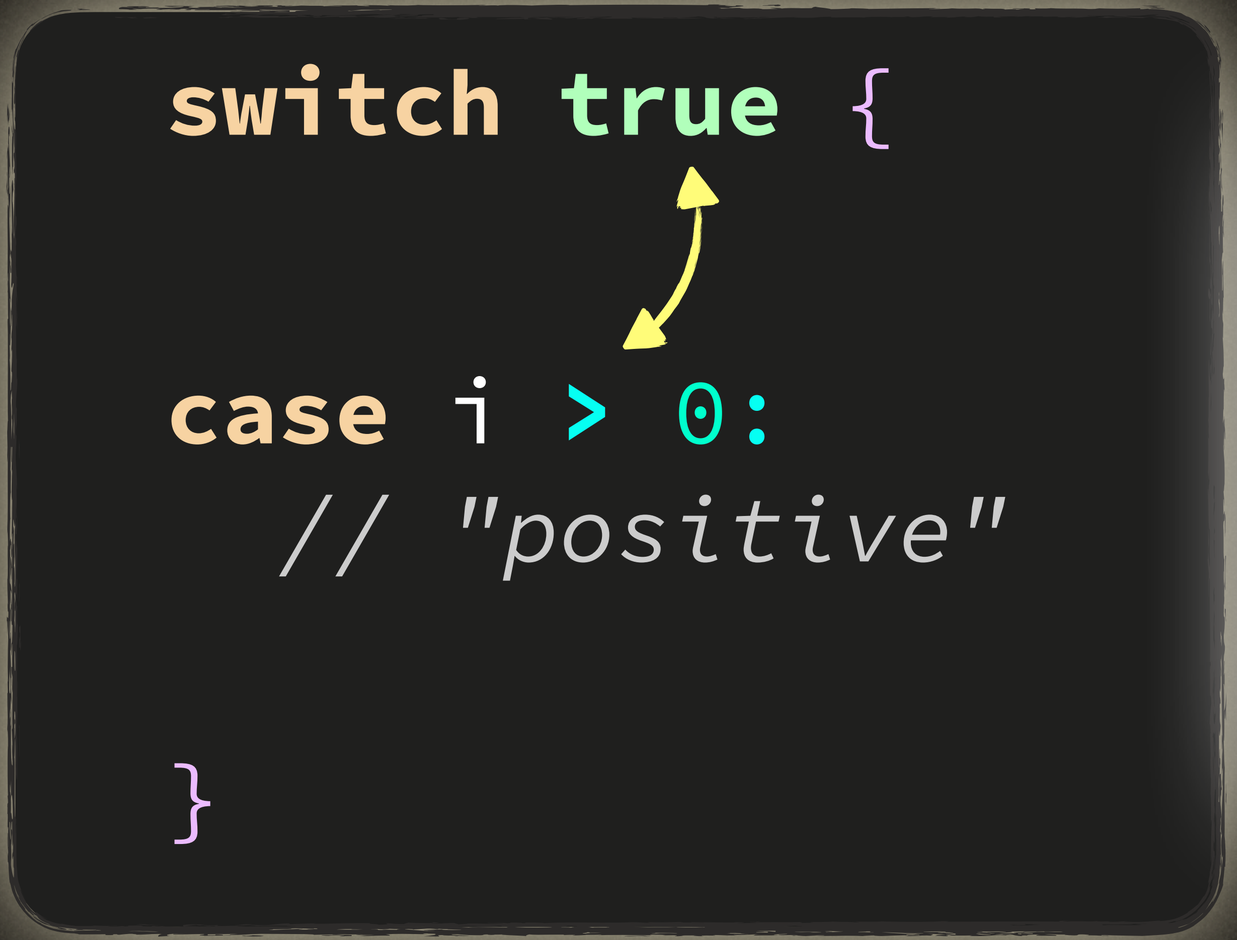
```
switch true {

case i > 0:
   // "positive"

}
```

BOOL EXPRESSIONS

Remember: Case conditions' types should match to condition expression's type



Both are bool expressions

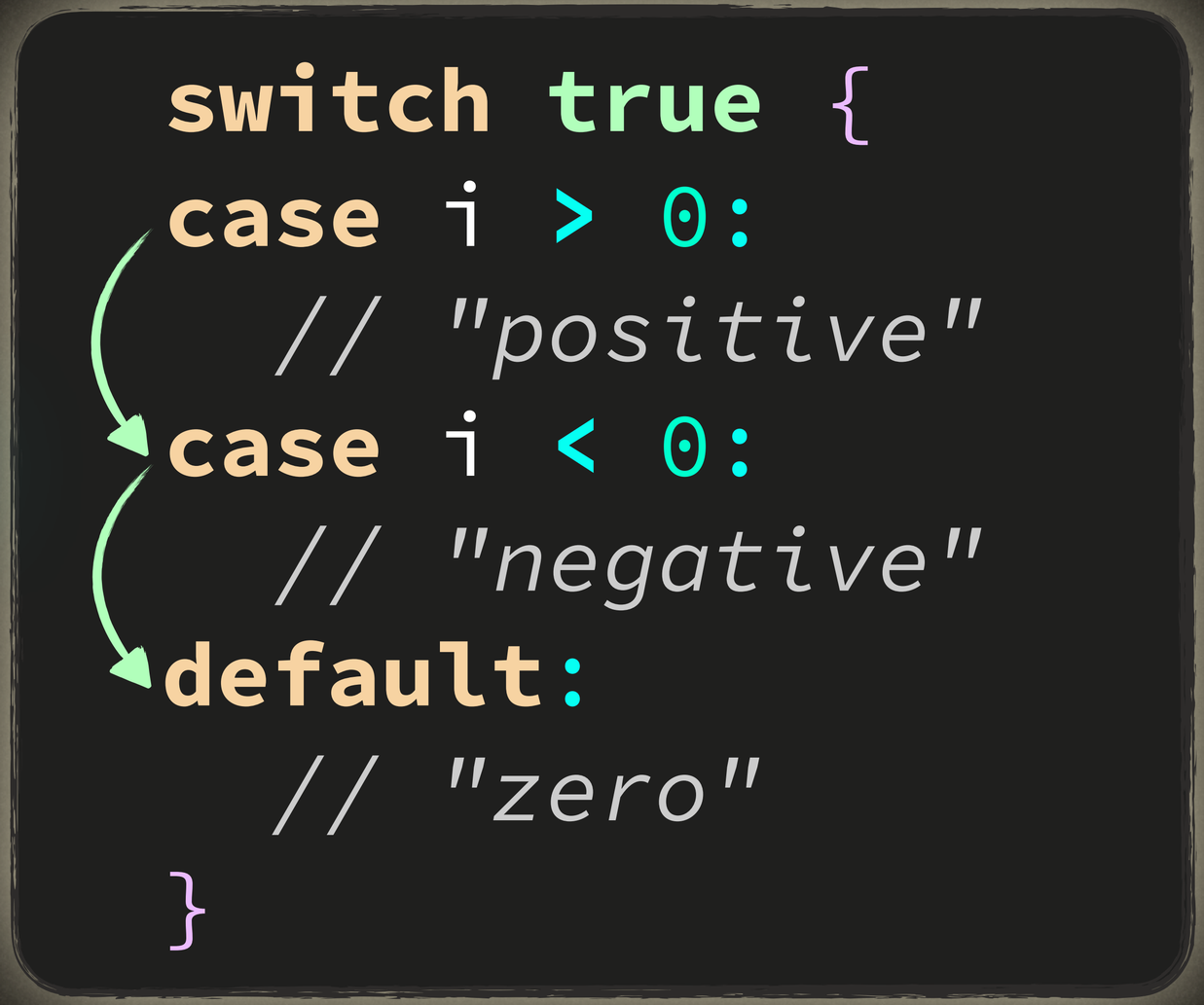
They return bool values

Their type is bool

BOOL EXPRESSIONS

The first case condition which yields true will be executed

Go executes a switch statement from top to bottom



```
switch true {
case i > 0:
    // "positive"
case i < 0:
    // "negative"
default:
    // "zero"
}
```

BOOL EXPRESSIONS

Switch condition is by default true

```
switch {
case i > 0:
   // "positive"
case i < 0:
   // "negative"
default:
   // "zero"
}
```

FALLTHROUGH STATEMENT

Fallthroughs to the next case clause unconditionally

FALLTHROUGH STATEMENT

Let's see an example first without fallthrough statements

```
i := 142

switch {
case i > 100:
   fmt.Print("big positive number")
case i > 0:
   fmt.Print("positive number")
default:
   fmt.Print("number")
}
```

```
$ go run main.go
big positive number
```

FALLTHROUGH STATEMENT

Let's see an example first without fallthrough statements

```
i := 142

switch {
case i > 100:
   fmt.Print("big positive number")
case i > 0:
   fmt.Print("positive number")
default:
   fmt.Print("number")
}
```

```
$ go run main.go
big positive number
```

FALLTHROUGH STATEMENT

Executes the next clause without checking for its condition

```go
i := 142

switch {
case i > 100:
    fmt.Print("big ")
    fallthrough
case i > 0:
    fmt.Print("positive ")
    fallthrough
default:
    fmt.Print("number")
}
```

```
$ go run main.go
big positive number
```

SWITCH STATEMENT

Congrats! Now you know how to use expressions with a switch statement!

SHORT SWITCH

Similar to short if statement

SHORT SWITCH

Previously...

```
i := 10

switch {
case i > 0:
   // "positive"
case i < 0:
   // "negative"
default:
   // "zero"
}
```

SHORT SWITCH

Simple statement runs once and first

```
switch i := 10; true {
case i > 0:
  // "positive"
case i < 0:
  // "negative"
default:
  // "zero"
}
```

SHORT SWITCH

Simple statement runs once and first

simple statement

```
i := 10
i = 10
i++
i--
```

any expression

func call() etc...

...

```
switch i := 10; true {
case i > 0:
  // "positive"
case i < 0:
  // "negative"
default:
  // "zero"
}
```

SEPARATOR

The separator can only be used once

separator
separates
the simple statement
and
the switch condition

```
switch i := 10; true {
case i > 0:
   // "positive"
case i < 0:
   // "negative"
default:
   // "zero"
}
```

SWITCH'S CONDITION

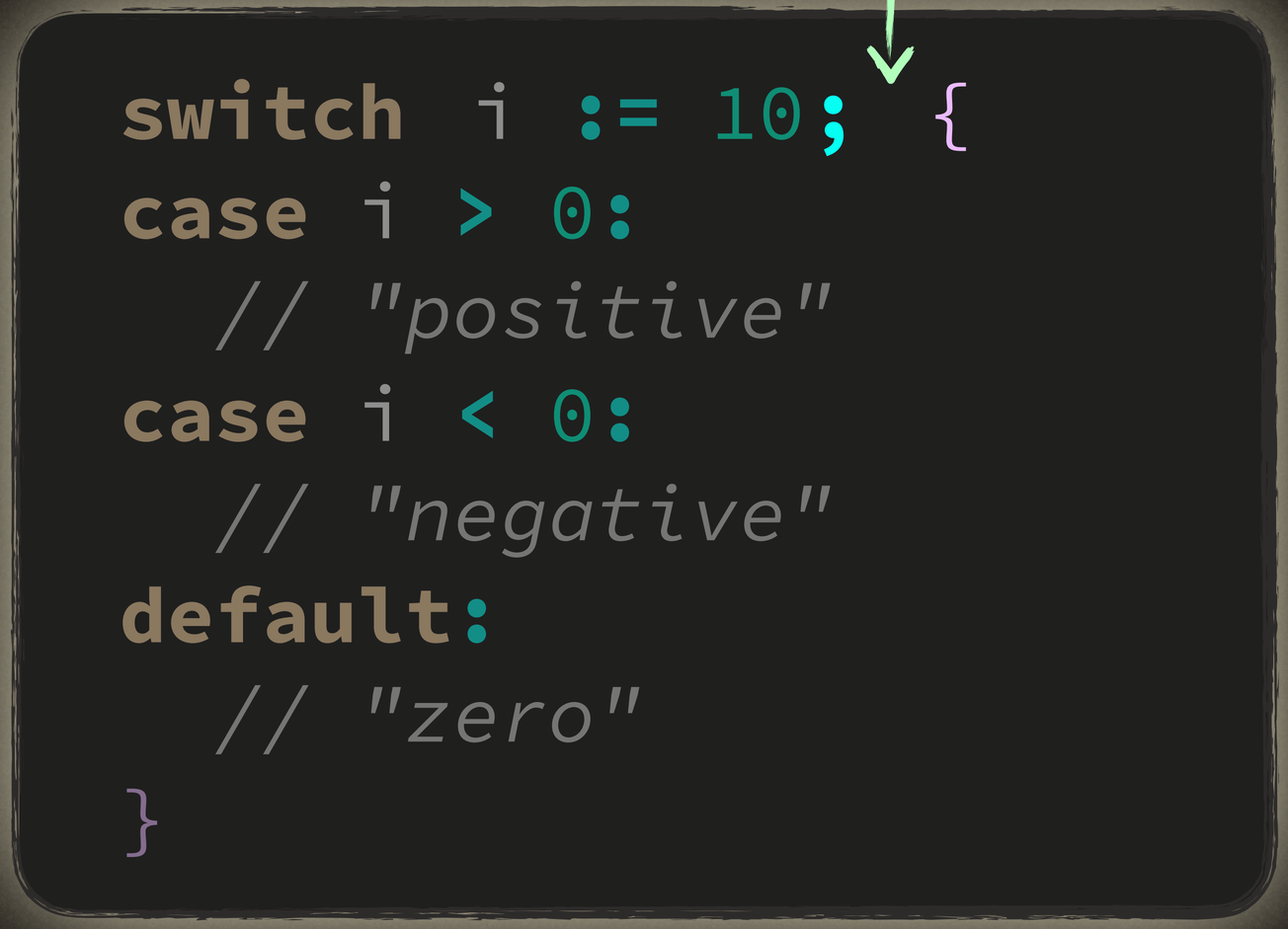
You need type the switch's condition next to the separator

switch's condition

```
switch i := 10; true {
case i > 0:
  // "positive"
case i < 0:
  // "negative"
default:
  // "zero"
}
```

SWITCH'S CONDITION

Switch condition is by default true



```
switch i := 10; {
case i > 0:
   // "positive"
case i < 0:
   // "negative"
default:
   // "zero"
}
```

SWITCH STATEMENT

Congrats! Now you know how to use the Short Switch!

PARTS of a DAY

Challenge

TIME PACKAGE

time

allows you to work with time

`time.Now()`

*Returns the **current** time as **Time***

`Time`

*Contains **methods** for manipulating **Time***

`Hour``()`

*Gets the **Hour** as **int** from the **Time** instance in 0 - 23 range*

```
time.Now().Hour()
```

Returns the current hour

`switch`

You should only use a switch statement

(Preferably use a short switch)

summary

top to bottom

checks first →

checks last →

last resort →

```
switch a {
case 1:
case 3:
case 5:
case 7:
default:
}
```

automatic break

```
switch a {
case 1:
  fallthrough
case 5:
default:
}
```

bool expressions

```
switch /* true */ {
case true && true:
case !true:
}
```

switch vs if

```
if a == 1 || a == 2 {
} else if a == 3 || a == 4 {
} else if a == 5 || a == 6 {
} else if a == 7 || a == 8 {
} else {
}
```

```
switch a {
case 1, 2:
case 3, 4:
case 5, 6:
case 7, 8:
default:
}
```

→ *Each condition checked left to right*

SWITCH STATEMENT

Congrats! Now you know everything about the switch statement!

PART III

Fundamentals

Control Flow